CORPUS WORK WITH PC BETA

A Presentation

Benny Brodda,
University of Stockholm
Dept. for Comp. Ling
S-106 91 Stockholm, Sweden
benny@com.qz.se

0. Abstract.

PC Beta is a PC oriented tool for corpus work in this term's broadest possible sense. With PC Beta one can prepare texts for corpus work, e.g. standardize texts in different ways (very important when texts from different sources together will constitute a corpus), one can process texts, and one can analyze texts. Making ordinary concordances and similar things with PC Beta is, of course, very simple, and, in fact, PC Beta gives "concordance making" a new dimension. One can perform morphological analyses, one can use PC Beta as a "tagger", i.e. provide the words with different kinds of tags. In all, PC Beta is a versatile program, and it is in many cases the only program needed (together with functions belonging to the MS/PC-DOS operative system) for pursuing a complete corpus project.

The program's main distinctive feature is simplicity: it is rule controlled, and the rules adhere to a format that any linguist can learn to understand very quickly. But beware, in spite of its innocent appearence the program is a little tiger.

1. The Programming System

1.1 Background.

PC Beta has its origin in a program called Beta, which the author developed during the years 1974–78. Beta was then specifically tied to a trade mark management project, in which it was used for morpho/phonological parsing of trade marks at the word level. Beta was then optimized for surface-oriented analysis and processing, and it turned out to be useful for morpho/phonological parsing of that type for "ordinary" language as well (cf. Brodda & Karlsson, 1981, and Källgren, 1982). Even if experience has shown that Beta can be used for much more advanced types of analyses (cf. Brodda, 1983, Källgren, 1984a, Källgren, 1984b and Brodda, 1988), it is still in surface oriented analysis (not necessarily confined to the word level) that its virtues become most apparent, although it may be used also for traditional parsing, traditional morphological analysis, etc.

During the years 1980–88 further development of the program was done on a DEC 10 computer, and a version called BetaText eventually emerged, which had several features specifically aimed at facilitating "corpus work", i.e. the processing and/or analysis of text corpora of the Brown, London-Oslo-Bergen, London-Lund types (cf. Erman, 1987, and Brodda, 1988). It is experiences with BetaText that lie behind the development of PC Beta (cf. Malkior & Carlvik, 1990).

One very important feature of PC Beta is that it takes ordinary text files as input and yields ordinary text files as output; PC Beta is a text processing system, not a database system. When you work with PC Beta, 1 Mb text requires 1 Mb disk space. This means that one can work with quite susbstantial text corpora on a standard PC/XT or AT with 20Mb disk, and still have space for auxiliary programs, sorting etc.; PC Beta itself and its auxiliary

files takes less than 35kb of diskspace, and rule files are typically only a few kb each, although they may presently be as large as 50 kb.

Now a few words on hardware requirements. The version of PC Beta presented here will run on any IBM/PC compatible computer, and, in fact, there is astonishingly much you can do with PC Beta on an ordinary PC with only two floppy disks. Working with a hard disk is, of course, easier, and is necessary if you need to work with larger texts than 250kb or so. PC Beta becomes more "snappy" if you have a PC with a 286 processor, not to mention one with a 386 processor.

During the spring of 1990 we will have a Macintosh version ready, to begin with only as a direct transfer from the PC version (by simply recompiling the source code on a Mac), but later we hope to get a "real" MAC-Beta, a version that will be programmed more in the MacIntosh fashion.

1.2. What is PC Beta?

Technically speaking, PC Beta is a straightforward production system in the sense of Rosner, 1983. Whenever PC Beta is used, its actions are completely controlled by a set of rules, so called productions, which the user supplies (a production is, in short, a rewrite rule which may be more complex than rewrite rules of the type linguists are used to; cf. e.g. Smullyan, 1961;). "Completely" means here exactly what it says; there are practically no built in actions in the program and the user has full control over what the program should do with the text it processes. Furthermore, the rules conform to a format that any linguist can learn to under stand and write in quite a short time, thus making himself his own computational linguist.

Before describing the rules and their functions, let me mention briefly that when designing the Beta rule format, I had the following goal in mind: simple things should be simple to do, but one should also have the possibility to do complex things when the application so demands. There are numerous examples of rule systems (a "rule system" = the actual rules plus accompanying declarations and specifications) that are almost ridiculously simple. A rule system, for instance, for producing a KWIC concordance of all the words in an arbitrary text requires in principle one rule only (plus a few specifications of the input and output formats). A rule system for rinsing a text from control characters can even be of length zero. A rule system for adding line numbers to a text needs only one specification of the type "NUM = 5", informing the program that a line number field (of width five in this case) is to be added in front of every record in the output file. One can learn to write rule systems for simple tasks like the ones mentioned in a few hours. But using PC Beta is like playing chess, one can learn the rules of the game in a couple of hours, and with some experience one can become quite a good at it, but it still takes a lot of experience and imagination to become a master.

Fortunately, every new user of PC Beta does not have to "invent the wheel". In the course of time quite substantial experience in using the program has been made. "Brodda (1990)", referred to several times in this article is, in fact, a straightforward "compendium", exclusively dedicated to the use of PC Beta in corpus work and will contain detailed descriptions of a host of rule systems, all useful in practical corpus work activities. It will also provide a lot of hints on what one has to think about when pursuing a corpus project, both in general and with PC Beta specifically.

1.3. How does the program work?

The computational setup in PC Beta is the following: PC Beta reads one record (cf. section 4, below) at a time from the given input file and places it in an internal working storage, WS. An internal state variable is given an initial value = 1 and a cursor is – metaphorically – placed at the leftmost end of WS. As long as no rule is applicable at the current position of the cursor, this is moved rightwards one step at a time until, eventually, an applicable rule is found. If this happens, the rule is applied (the content of WS is changed, for instance), upon which the cursor is moved to a position defined by that rule. From there new applicable rules are searched for, until – hopefully – the cursor moves

outside WS to the right, and the processing of the current record is over. The current content of WS is then sent to the chosen output channel and a new record is brought in from the input file, and so on until the input file is is emptied.

The rule file that controls the actions of PC Beta contains primarily the rules themselves, but also some necessary declarations, essentially definitions of various state and character sets, and for mat descriptions, such as information about whether there is a line header field in the input file and, if so, how wide it is. ("Line header" a "line ID" in line-initial position; cf. section 2.1). The main part is, of course, contained in the rules section of the program, and I shall now proceed to describe briefly how PC Beta rules are constructed.

Theoretically – in practice they look differently – a PC Beta rule is a 7-tuple:

(1) < Observed string, Context condition(s), State condition,

, Resulting string, Resulting state, Move, Resulting actions >

The first three elements in (1) define rule conditions: "Observed string" must be an explicitely given string (cf. section 5) and the condition is fulfilled if an instance of "Observed string" is found at the current position of the cursor. "Context condition" breaks down to two subconditions, one left context condition (of the ob served string) and one right context condition (of the same string). "State condition" is a condition on the internal state. The last four elements in (1) above define what happens when the rule condi tions are fulfilled. "Resulting string" is a string that replaces (the instance of) the observed sting in WS. "Resulting state" defines the new value of the internal state. "Move" is a directive of where in WS to put the cursor after the application of the rule. This position is typically defined relative to the newly inserted "resul ting string", but the cursor may also be directed to other places in the string under processing.

The component "Resulting actions" in (1) is extremely important in corpus work applications. In PC Beta there is a possibility to define specific sets of states with reserved names, and a specific action is tied to each such set; whenever the internal state happens to become a member of such a set, the corresponding action is in voked. Such internal states are collectively referred to as "action states" (cf. Brodda, 1988). Now, some of the actions that can be invoked in this way are typical "things" one wants to do in typical corpus applications: move an observed string out to a KWOC-field, print the current record when something interesting has been found – this is excerption – and perhaps format the output in such a way that the position of the cursor always appears in a predefined print po sition – this is how KWIC concordances are obtained – and so on.

Before leaving this topic I think there is a theoretical point calling for a remark here. As anybody with some minimum knowledge of mathematical linguistics can see, the rule format (1) is a kind of generalization of Turing machine rules, which implies that the PC Beta programming system in principle is a general Turing machine. Thus, it is a trivial consequence that with PC Beta one can achieve whatever text warping one can ever dream up. There is no other limit than imagination and computer space. Thus, when I claim that one can do complex things with PC Beta it is, sort of, a very trivial remark.

What I mean is that one can do quite many things, some rather complex, under the heading "Computational Linguistics" in a natural way. The rule format (as well as the whole setup) is tuned to be efficient for typical applications in that area, and with special attention to surface oriented analysis. This format has been arrived at after years of experimenting and actual testing in true situations; in principle I began with a system that was much more ambitiuos than the present and then I primarily sacrified features that turned out to be unnecessary and/or never used. Some other features have been modified and a few other added (but very conservatively). What is left is a kind of basic tool for computational linguistics.

1.4. *What is a record?*

As mentioned above, PC Beta is record-oriented: it reads in and processes one record at a time. Now, what is a re cord? In computer connections text lines usually constitute the basic physical records when a text is processed, but, with the exception of poetry, text lines do not constitute very natural linguistic units, and therefore one has to have means to overrule this default record definition. In PC Beta we have adopted a simple, yet very effective, way to accomplish this. The logical records one can define are typically words, sentences or paragraphs; larger chunks than normal paragraphs can usually not be kept in the working storage of the present version of PC Beta. (The limit is around 3500 characters.)

When line headers are present, each record is associated with the line header of the line where the record begins, and this line header is then the one that usually appears when the record is output.

As I mentioned earlier, the internal state is by default reset to 1 whenever a new record is brought into WS. This implies that each logical record is processed as if in isolation. This default is, how ever, easily overruled, and then the value of the internal state is kept as it was from the preceding record, when a new record is brought in. In some sense, PC Beta considers the whole text as one logical record when run in this mode.

1.5. *More on PC Beta rules.*

In section 3, above, I described PC Beta rules from an abstract point of view. Now, PC Beta rules are not abstract entities, they are very concrete: they contain an ordinary rewrite component of the type "X - Y" where X and Y denote strings ("X is rewritten as Y"), and these strings must be explicit. Thus, an alleged phonological rule of the type: "V - + (back)/..." is meaningless unless the symbol "V" itself (and not only objects classified as "V"s) appears in the text. Furthermore, in this case a letter V appearing in the right environments is simply rewritten as the string " + (back)", which perhaps is not exactly what a phonological rule of the type mentioned would mean.

The "concretism" is not a shortcoming of PC Beta, it is a deliberately chosen feature. One reason for this choice is that all such built-in properties delimit generality (otherwise: which action is the program supposed to take if you actually want to rewrite "V" as " + (back)"?). Another reason is efficiency: if the program in all situations has to check whether the user actually means what a rule says, or whether there is an implicit category involved that is going to be changed in some abstract way, then it will take time. A third —and the main— reason is that I am personally a concrete linguist, I simply think that rules in linguistics should be concrete as far as possible.

Of course I have to admit that there are instances when it would be convenient to refer to, say, any vowel simply as "V" in the re write part of a rule. Are there ways to achieve this in PC Beta? Yes, there are. In Brodda & Karlsson 1981 it is shown that such abstrac tions are easily taken care of by meta rules, Beta-rules that expand abstract categories like the ones mentioned and also modify rules in other ways. A slightly more complex example of this type will be described below (section 2.3).

1.6. *Rule conditions and the internal state.*

Each rule contains two context conditions, one for the left context and one for the right context, plus one condition on the current internal state. All these three conditions are evaluated in a similar way, and all three must be fulfilled for the rule to be applicable; a superordinate condition is, of course, that the "observed string" actually is located at the current position of the cursor.

The context and state conditions appear in the rules as the names of three sets, two character sets and one "state" set. The context conditions are fulfilled if the character to the left of (the in stance of) the observed string belongs to the set denoted by the left-context condition, and, similarly, the character to the right of the observed string belongs to the set denoted by the right-context condition, these sets being defined under the heading CHARSET ("character sets") in the actual rule file.

The internal state, IS, is an internal variable that can take arbitrary positive integers as values. The internal state is intitialized to 1 when the processing begins, and usually again when a new record is brought into the working storage. From there on the internal state is successively updated through the applications of rules, and by having a condition on this internal state in each rule one can achieve logical chaining of whole sets of rules. Roughly one can say that the context conditions take care of the immediate environment whereas the internal state condition embodies more abstract and arbitrarily complex conditions on the structure in which the observed string appears.

The state condition in a rule is again just a name, now referring to a set of positive integers (i.e. possible states) defined under the heading STATESET ("state sets") in the rule file, and the condition is fulfilled if the current internal state is a member of that set. To understand this way of evaluating state conditions is the whole key to understanding PC Beta programming.

A critic may wonder why we do not allow more complex (near) context conditions than just conditions on the immediate left and right characters. The reason is efficiency. Testing a character for membership in a character set (or a state for membership in a state set) is done in a very fast and simple way, whereas testing a string for membership in a string set requires some sort of lexicon lookup procedure, which is, generally speaking, a comparatively more complex operation. – OK, but if a rule actually requires specific strings in its context conditions, how do you handle that? – Simple, move the cursor around a little and establish the context conditions as specific changes in the internal state via the applications of rules. – But isn't that just a makeshift? Don't you need lexicon lookups for establishing certain contexts as "observed strings"?

Undoubtedly there is a point there, so in the next release of PC Beta (due to appear, about a year or so from now) we will probably allow a third heading, STRINGSET, under which arbitrary sets of strings may be defined, the name of which may then be used as left or right hand conditions in rules as alternatives to character conditions. The reason why this is not implemented already is, primarily, that in most cases character contexts are perfectly sufficient, and, besides, it is not entirely clear to us what conventions this string set feature should follow in all details, technically or theoretically; it will take some experimenting to decide that.

REFERENCES

Brodda,B. & Karlsson,F. "An Experiment with Automatic Morphological Analysis of Finnish", *Publ. No. 7*, Department of Linguistics, University of Helsinki, 1981.

Brodda,B. "Problems with tagging - and a solution", *Nord. Journal of Linguistics*, **5**, 1982, pp. 93 – 116.

Brodda, B. "An Experiment with Heuristic Parsing of Swedish" in Karlsson, F. (ed.) *Papers from the 7th Scandinavian Conference of Linguistics*, University of Helsinki, 1983.

Brodda,B. "Tracing Turns in the London-Lund Corpus with BetaText" in *Literary and Linguistic Computing*, **Vol. 3, No. 2**, 1988.

Brodda,B. *"Corpus Work with PC Beta"*, Inst. of Linguistics, University of Stockholm, 1990, forthcoming.

Erman,B. "Pragmatic Expressions in English" (dissertation), Stockholm Studies in English, *Acta Univ. Stockholmiensis*, 1987.

Källgren,G. "FINVX - a System for the Bakwards Application of Finnish Consonant Gradation Rules", *PILUS* **No. 42**, Inst. of Linguistics, University of Stockholm, 1982.

Källgren,G. "Automatisk Excerpering av substantiv ur löpande text", *IRI-rapport 1984:1*, Inst. för Rättsinformatik, University of Stockholm, 1984(a).

Källgren,G. "HP, a Heuristic Finite State Parser based on Morpho logy", in Sågvall-Hein (ed.) *"De Nord. Datalingvistikdagarna 1983"*, Uppsala University, Uppsala, 1984(b).

Malkior, S. & Carlvik, M. *PC Beta Reference*. Institute of Linguistics, University of Stockholm, 1990.

Rosner,M. "Production Systems" in *"Parsing Natural Languages"*, M. King (ed.), Academic Press, 1983

Smullyan, R.M. *"Theory of Formal Systems"*, Annals of Math. Studies, New York, 1961.